UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 10/731,839 | 12/09/2003 | Tatsushi Inagaki | JP920030039US1 | 4418 |

25259 7590 11/09/2006

IBM CORPORATION
3039 CORNWALLIS RD.
DEPT. T81 / B503, PO BOX 12195
REASEARCH TRIANGLE PARK, NC 27709

| EXAMINER | |
|---|---|
| NGUYEN, PHILLIP H | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2191 | |

DATE MAILED: 11/09/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

| Office Action Summary | Application No. | Applicant(s) | |
|---|---|---|---|
| | 10/731,839 | INAGAKI ET AL. | |
| | Examiner | Art Unit | |
| | Phillip H. Nguyen | 2193 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR.1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on *09 December 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *1-11* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-11* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *09 December 2003* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *20040407*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

## DETAILED ACTION

1. This action is in response to the original filing of December 9, 2003. Claims 1-11 are pending and have been considered below.

### *Drawings*

2. The drawings are objected to because Figure 1, item 105, 110, and 120 are needed to correct because unit 110 and 120 are obtaining the order constraint graph stored in the unit 105, the arrows must pointed to the other direction instead. Corrected drawing sheets in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application. Any amended replacement drawing sheet should include all of the figures appearing on the immediate prior version of the sheet, even if only one figure is being amended. The figure or figure number of an amended drawing should not be labeled as "amended." If a drawing figure is to be canceled, the appropriate figure must be removed from the replacement sheet, and where necessary, the remaining figures must be renumbered and appropriate changes made to the brief description of the several views of the drawings for consistency. Additional replacement sheets may be necessary to show the renumbering of the remaining figures. Each drawing sheet submitted after the filing date of an application must be labeled in the top margin as either "Replacement Sheet" or "New Sheet" pursuant to 37 CFR 1.121(d). If the changes are not accepted by the examiner, the applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

Figures 3-5, and 7 should be designated by a legend such as --Prior Art-- because only that which is old is illustrated. See MPEP § 608.02(g). Corrected drawings in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application. The replacement sheet(s) should be labeled "Replacement Sheet" in the page header (as per 37 CFR 1.84(c)) so as not to obstruct any portion of the drawing figures. If the changes are not accepted by the examiner, the applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

***Specification***

3. The abstract of the disclosure is objected to because it contains more than 150 words. Correction is required. See MPEP § 608.01(b).

***Claim Rejections - 35 USC § 101***

4. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Claims 7- 9 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Claim 1 is non-statutory because the language of this claim is directed to software, per se, lacking of storage on a medium, which enables any underlying functionality to occur. Claims 8-9 are directly dependent on claim 1, and therefore, are rejected under the same reason set for in claim 1.

***Claim Rejections - 35 USC § 102***

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 1-11 are rejected under 35 U.S.C. 102(b) as being anticipated by R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao, "Minimum Register Instruction Sequence Problem: Revisiting Optimal Code Generation for DAGs", April 2001.

Claim 1: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose a compiler that optimizes a program to be compiled by changing the execution order of instructions in the program, the compiler comprises:

a. an order constraint information obtaining unit that obtains order constraint information indicating order constraints defined among a plurality of instructions in the program ("a data dependence graph" see page 2, Col 1, paragraph 8, under "Motivating Example"), the order constraints defining the order in which the instructions should be executed;

b. an order determination unit that sequentially determines the execution order for each of the plurality of instructions based on the order constraint information (see page 2, Figure 1(c), under "Motivating Example");

c. an unit for analyzing the number of registers that analyzes the number of required registers, which is the number of registers that will be required when the instructions are executed ("minimum register requirement is three", see page 2, Col 2, paragraph 1, under "Motivating Example");

d. an instruction detection unit that detects a combination of two instructions ("instructions b and e", see page 3, under "Overview of Our Approach", Figure 2(b)), in which one instruction is a determined order instruction ("The definition of the lineage L1={a, b, f, h} has been created. Therefore, b is determined order instruction" see page 3, under "Overview of Our Approach") for which the execution order has been determined by the order determination unit the other instruction is an undetermined order instruction ("the definition of the lineage L1 = {a, b, f, h} does not include e, and therefore e is undetermined order instruction", see page 3, under "Overview of Our Approach", Figure 2(a)) for which the execution order has not been determined by the order determination unit and the order constraint information does not include a constraint that the one instruction should be executed before the other instruction ("there is no constraint order between instructions b and e", see page 3, Figure 1(a), under "Overview of Our Approach"); and

e. an order determination reprocessing unit that, when the number of required registers exceeds a predetermined number ("in according to the Figure 2(b), the number of required registers allocate for storing the results of instructions a, c, d, and e as one, two, three, and four, respectively. Therefore, three is a predetermined number of registers has been exceeded", see page 3, Figure 1(b), under "Overview of Our

Approach"), changes the state of the one instruction into the state in which the execution order has not been determined ("instruction e is undetermined order instruction") and causes the executed next to the other instruction ("There is an order constraint goes from instruction e to instruction b", see page 3, Figure 2(b), under "Overview of Our Approach").

Claim 2: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler as in claim 1 above, and further disclose the instruction detection unit detects an instruction that releases a register as the other instruction ("in according to Figure 2(b), instruction g performs processing using the results of the instruction e and stores the processing result in the register that has been storing the result of e. Therefore, it releases a register, see page 3, Figure 2(b), under "Overview of Our Approach"), and an instruction that requires a new register allocated to it as the one instruction ("in according to Figure 2(b), instruction a requires a new register allocated to it for storing the result", see page 3, Figure 2(b), under "Overview of Our Approach").

Claim 3: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler as in claim 1 above, and further disclose the instruction detection unit detects an instruction that releases a register as the other instruction ("in according to Figure 2(b), instruction g releases register, see page 3, Figure 2(b), under "Overview of Our Approach"), and an instruction to be executed before a determined order instruction that requires a new register allocated to it as the on instruction ("the definition

of lineage L1={a, b, f, h}, does not include instruction e, and therefore, instruction e is undetermined order instruction. In according to Figure 2(b), there is a order constraint goes from instruction e to instruction b, and therefore, e is executed before to b" see page 3, under "Overview of Our Approach", Figure 2(b), and "instruction e is required a new register allocated to it for storing its result"), and changes the state of all instructions to be executed after the determined order instruction in the order constraint information into the state in which the execution order has not been determined ("the execution order for instruction e has not been determined" see page 3, Figure 2(b), under "Overview of Our Approach").

Claim 4: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler as in claim 1 above, and further disclose a plurality of combinations of the one instructions and the other instruction are detected by the instruction detection unit ("L1=(a, b, f, h), L2=(c, f), L3=(e, g, h), L4=(d, g)", see page 3, Figure 2(c), under "Overview of Our Approach"), the order determination reprocessing unit selects from the plurality of combinations a combination that minimizes the sum of the depth of order constraint from a start point of the program to the other instruction and the depth of order constraint from the one instruction to an end point of the program ("we choose a descendent node with the smallest height (depth)", see page 4, paragraph 1), the order determination reprocessing unit causes the order determination unit to determine the execution order using the other instruction and the one instruction

included in the selected combination ("L4={d, g}, in this case, d is one instruction and g is other instruction", see page 3, Figure 2(c), under "Overview of Our Approach")

Claim 5: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler as in claim 1 above, and further disclose when the number of required registers exceeds the predetermined number ("in according to the Figure 2(b), the number of required registers allocate for storing the results of instructions a, c, d, and e as one, two, three, and four, respectively. Therefore, three is a predetermined number of registers has been exceeded", see page 3, Figure 2(b), under "Overview of Our Approach"), the order determination reprocessing unit adds an order constraint that the determined order instruction should be executed next to the undetermined order instruction to the order constraint information ("There is an order constraint goes from instruction e, which is undetermined order instruction, to instruction b, which is determined order instruction." see page 3, Figure 2(b), under "Overview of Our Approach"), and thereby causes the order determination unit to determine the execution order so that the determined order instruction is executed next to the undetermined order instruction ("In according to Figure 2(b), instruction e is executed next to instruction b", see page 2, Figure 2(b), under "Overview of Our Approach").

Claim 6: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler as in claim 5 above, and further disclose the order constraint information obtaining unit obtains, as the order constraint information ("a data

dependence graph with instruction sequences" see page 2, under "Motivating Example"), an order constraint graph that represents each instruction in the program as a node and order constraint under which a plurality of instructions should be executed as directed edges ("DDG graph" see page 2, Figure 1(a), under "Motivating Example"), the order determination unit determines the execution order based on the order constraint graph so that an instruction represented by a start node of a directed edge is executed before an instruction represented by an end node of the directed edge ("two possible instruction sequences for DDG graph", see page 2, Figure 1(b-c), under "Motivating Example"), the instruction detection unit detects that the order constraint information does not include an order constraint that the one instruction should be executed before the other instruction by detecting a combination of two instructions in which a node representing the one instruction cannot reach a node representing the other instruction on the order constraint graph ("In according to Figure 2(a), node e cannot reach node b on the order constraint graph", see page 3, Figure 2(a), under "Overview of Our Approach"), and the order determination reprocessing unit adds an order constraint that the other instruction should be executed next to the one instructions to the order constraint information by generating a directed edge from the node representing the undetermined order instruction to the node representing the other instruction ("In according to Figure 2(b), after the definition of lineage L1={a, b, f, h} has been created, there is an order constraint goes from node e to node b" see page 3, Figure 2(b), under "Overview of Our Approach").

Claim 7: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose a compiler program for causing a computer to function as a compiler that optimizes a program to be compiled by changing the execution order of instructions in the program, wherein the compiler program causes the computer to function as:

a. an order constraint information obtaining unit that obtains order constraint information indicating order constraints defined among a plurality of instructions in the program (see page 2, Figure 1, under "Motivating Example"), the order constraints defining the order in which the instructions should be executed;

b. an order determination unit that sequentially determines the execution order for each of the plurality of instructions based on the order constraint information (see page 2, Figure 1(c), under "Motivating Example");

c. an unit for analyzing the number of registers that analyzes the number of required registers, which is the number of registers that will be required when the instructions are executed ("minimum register requirement is three", see page 2, Col 2, paragraph 1, under "Motivating Example");

d. an instruction detection unit that detects a combination of two instructions ("instructions b and e", see page 3, Figure 2(b), under "Overview of Our Approach"), in which one instruction is a determined order instruction ("The definition of the lineage L1={a, b, f, h} has been created. Therefore, b is determined order instruction" see page 3, under "Overview of Our Approach") for which the execution order has been determined by the order determination unit the other instruction is an undetermined order instruction ("the definition of the lineage L1 = {a, b, f, h} does not include e, and

therefore e is undetermined order instruction", see page 3, Figure 2(a), under "Overview of Our Approach") for which the execution order has not been determined by the order determination unit and the order constraint information does not include a constraint that the one instruction should be executed before the other instruction ("there is no constraint order between instructions b and e", see page 3, Figure 1(a), under "Overview of Our Approach"); and

e. an order determination reprocessing unit that, when the number of required registers exceeds a predetermined number ("in according to the Figure 2(b), the number of required registers allocate for storing the results of instructions a, c, d, and e as one, two, three, and four, respectively. Therefore, three is a predetermined number of registers has been exceeded", see page 3, Figure 2(b), under "Overview of Our Approach"), changes the state of the one instruction into the state in which the execution order has not been determined ("instruction e is undetermined order instruction") and causes the executed next to the other instruction ("There is an order constraint goes from instruction e to instruction b", see page 3, Figure 2(b), under "Overview of Our Approach").

Claim 8: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler program as in claim 7 above, and further disclose the instruction detection unit detects an instruction that releases a register as the other instruction ("in according to Figure 2(b), instruction g performs processing using the results of the instruction e and stores the processing result in the register that has been storing the

result of e. Therefore, it releases a register, see page 3, under "Overview of Our Approach", Figure 2(b)), and an instruction that requires a new register allocated to it as the one instruction ("in according to Figure 2(b), instruction a requires a new register allocated to it for storing the result", see page 3, under "Overview of Our Approach", Figure 2(b)).

Claim 9: R.Govindarajan, H Yang, J. N. Amaral, C Zhang, and G. R. Gao disclose the compiler program as in claim 7 above, and further disclose when the number of required registers exceeds the predetermined number ("in according to the Figure 2(b), the number of required registers allocate for storing the results of instructions a, c, d, and e as one, two, three, and four, respectively. Therefore, three is a predetermined number of registers has been exceeded", see page 3, under "Overview of Our Approach", Figure 2(b)), the order determination reprocessing unit adds an order constraint that the determined order instruction should be executed next to the undetermined order instruction to the order constraint information ("There is an order constraint goes from instruction e, which is undetermined order instruction, to instruction b, which is determined order instruction." see page 3, under "Overview of Our Approach", Figure 2(b)), and thereby causes the order determination unit to determine the execution order so that the determined order instruction is executed next to the undetermined order instruction ("In according to Figure 2(b), instruction e is executed next to instruction b", see page 2, under "Overview of Our Approach", Figure 2(b)).

Claim 10: A recording medium with the compiler program according to claim 7 recorded on it. It recites the limitations as recited in claim 7 above, and therefore, has been addressed in connection with the rejection of claim 7 above.

Claim 11: a compiling method for optimizing a program to be compiled by changing the execution order of instructions in the program with a computer, the method comprising:

a. obtaining order constraint information indicating order constraints defined among a plurality of instructions in the program ("a data dependence graph" see page 2, Col 1, paragraph 8, under "Motivating Example"), the order constraints defining the order in which the instructions should be executed;

b. sequentially determining the execution order for each of the plurality of instructions based on the order constraint information (see page 2, under "Motivating Example", Figure 1(c));

c. analyzing the number of registers that analyzes the number of required registers, which is the number of registers that will be required when the instructions are executed ("minimum register requirement is three", see page 2, Col 2, paragraph 1, under "Motivating Example");

d. detecting a combination of two instructions ("instructions b and e", see page 3, under "Overview of Our Approach", Figure 2(b)), in which one instruction is a determined order instruction ("The definition of the lineage L1={a, b, f, h} has been created. Therefore, b is determined order instruction" see page 3, under "Overview of

Our Approach") for which the execution order has been determined by the order determination unit the other instruction is an undetermined order instruction ("the definition of the lineage L1 = {a, b, f, h} does not include e, and therefore e is undetermined order instruction", see page 3, under "Overview of Our Approach", Figure 2(a)) for which the execution order has not been determined by the order determination unit and the order constraint information does not include a constraint that the one instruction should be executed before the other instruction ("there is no constraint order between instructions b and e", see page 3, under "Overview of Our Approach", Figure 1(a)); and

e. when the number of required registers exceeds a predetermined number ("in according to the Figure 2(b), the number of required registers allocate for storing the results of instructions a, c, d, and e as one, two, three, and four, respectively. Therefore, three is a predetermined number of registers has been exceeded", see page 3, under "Overview of Our Approach", Figure 2(b)), changes the state of the one instruction into the state in which the execution order has not been determined ("instruction e is undetermined order instruction") and causes the executed next to the other instruction ("There is an order constraint goes from instruction e to instruction b", see page 3, under "Overview of Our Approach", Figure 2(b)).

***Conclusion***

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Phillip H. Nguyen whose telephone number is (571)

270-1070. The examiner can normally be reached on Monday - Friday 10:00 AM - 3:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (571) 272-3719. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

PN
10/11/06

Kakali Chaki
Supervisory Patent Examiner

KAKALI CHAKI
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100